Санкт-Петербургский Государственный Университет
Информационных Технологий, Механики и Оптики

Кафедра проектирования компьютерных систем

Отчёт по лабораторным работам №1, №2
по дисциплине
"Основы системного программирования"

Проверил:

Работу выполнил:
Захаров Р. Н.
группа 3156

- Санкт-Петербург, 2010 -

Оглавление

1. Условия лабораторных работ

Задание к первой работе формулируется как:

30. Сформировать одномерный массив A = {a_i} (i = 1 .. 30), где a_i вычисляется по формулам:

$\mathbf{a_i = i * 2 - 5}$	$\mathbf{0 < i < 11}$
$\mathbf{a_i = (i + i / 3) / 2 * 5}$	$\mathbf{10 < i < 21}$
$\mathbf{a_i = 5 * i / (1 + i)}$	$\mathbf{20 < i < 31}$

Вторая лабораторная работа является продолжением первой - необходимо обработать результат выполнения первой программы - создать 2 массива, в которые поместить:

- в первый - элементы массива, содержащие цифру **5** в своём десятичном написании
- во второй - элементы массива, которые делятся нацело на **10**

1.1. Примечания

В качестве ОС, под которую писал лабораторные работы - Linux, используемый компилятор - NetwideASM(NASM) - практически единственный из распространённых под эту ОС, поддерживающий синтаксис Intel.

Компилируемые программы - 32х разрядные.

В качестве результата работы (в обоих заданиях это массивы) - вывод дампа из памяти в файл и для более наглядного вида в консоль.

В первой лабораторной использовалась только одна функция из библиотеки си - **printf**. Открывался файл, записывались данные, закрывался файл через функции вектора прерывания ядра - 80h.

Домашние задания лежат в папках HW1 и HW2 соответственно. В каждой папке (в т.ч. и корневой) есть makefile, который упрощает процесс сборки, запуска, и сравнения результатов

Первая лабораторная зависит от второй: использует её данные. Поэтому перед запуском второй, нужно убедиться, что дамп массива от результата работы первой есть, по отношению к **../HW1/dump_asm.bin**, иначе при запуске в консоль будет выведено сообщение об ошибке.

Для проверки корректности работы программ на ассемблере написал анлоги на си. Для сборки и выполнения их нужно выполнить **make run-test** в нужном дз. Каждая программа запишет дамп в файл(с приставной _c) в текущую папку.

2. Лабораторная работа №1

2.1. Логика работы

В главном цикле **fillup:** программы перебираются все числа из заданного диапазона, и в зависимости от текущего значения **i** (фактически в регистре **ecx**) принимается решение, переходить к обработке значения (к меткам **first:**, **second:**, **thrid:**) в зависимости от значения, по заданию.

Соответственно, в этих частях кода, происходит вычисление значения по формуле задания. Результат вычисления - в **eax**(хотя в действительности используется только **al**). После вычисления происходит переход к метке **fillup.endloop:** где вычисленное значение помещается в нужный элемент массива после вычисления его адреса.

После цикла, по метке **dodump:** происходит создание и сохранение(по метке **dodump.write:** массива в файл в бинарном виде(так, как лежит в памяти) и закрытие дескриптора (**dodump.closefile:**).

В конце происходит вывод на экран в цикле в удобочитаемой форме(по метке **output:**), где формируются аргументы для **printf**, куда входит извлечение элементов массива из памяти и приведение их от 8bit-signed к 32bit-signed.

Завершается работа кодом с меткой **sys_exit:** - происходит вывод функции ядра для выхода, с указанием результата - **0**(обычно означает, что завершилась успешно).

2.2. Код (листинг файла hw1.asm)

```
extern printf       ; the C function, to be called

section .data
    ar_len:     equ 31
    ar:         times ar_len db 0
    tmp:        db '  ', 10
    dumpfile:   db './dump_asm.bin', 0
    dbgline:    db 'ar[%d] = %d', 10, 0

section .text
    global _start


_start:         ; program start point

    mov ecx, ar_len
```

```
; Form array main loop
fillup:

    mov eax, ecx; copy

    cmp ecx, 31 ; more then 32?
    jg .endloop ; o_O

    cmp ecx, 20 ; 20 <= i <= 31 ?
    jg thrid

    cmp ecx, 10 ; 10 <= i < 20 ?
    jg second

    cmp ecx, 0  ; 0 <= i < 10 ?
    jg first

.endloop:       ; save calculated value in array node
    push eax

    mov eax, ecx
    mov dl, 1   ; ar - array of _DB_
    mul dl      ; in eax - real offset of element
    mov ebx, ar ; address of array
    add ebx, eax; finally, add base address of array

    pop eax     ; restore calculated value

    mov [ebx], al ; Save value to memory

    loop fillup ; Loop until ecx != 0


dodump:     ; make memory dump to file
    mov eax, 8          ; Syscall number for creat()
    mov ebx, dumpfile   ; Filename address
    mov ecx,00644Q      ; Permissions in octal (rw_rw_rw_)
    int 80h             ; Call the kernel

    test eax, eax       ; Lets make sure the file descriptor is valid
    js  sys_exit        ; Something gonna wrong...

.write:     ; write memory block to file
    mov ebx, eax        ; File handle was in eax
    mov eax, 4          ; Syscall number for write()
    mov ecx, ar + 1     ; Base address for ar
    mov edx, ar_len - 1 ; Length of array
    int 80h             ; Call kernel
```

```
.closefile: ; like fclose()
    mov eax, 6          ; sys_close
    int 80h             ; Call kernel (ebx already contains file descriptor)


output:
    mov edx, 1
out:
    push edx        ; Save counter

    mov eax, ar     ; Loading ar base
    add eax, edx    ; Add current offset
    mov eax, [eax]  ; Get 4 byte value at address
    and eax, 0xFF   ; Extract only lower byte

    test eax, 0x80  ; Test for high bit in al (sign)
    jz .args        ; There is no sign?

    or eax, ~0x7F   ; Inverse(set) 32-7 bits in sign number

.args:       ; form printf arguments
    push eax        ; First push - last argument. It's memory value
    push edx        ; Second push - is element number
    push dbgline    ; Address of string
    call printf     ; Call printf
    add esp, 12     ; pop stack 3 push times 4 bytes (cleanup)


    pop edx         ; Restore counter

    inc edx         ; Increase element number
    cmp edx, 31     ; It's a last element number?
    jne out         ; No, another iteration


sys_exit:   ; System call for exit
    mov eax, 1      ; Function number for exit program
    mov ebx, 0      ; Return code - it's okay (0)
    int 80h         ; Do kernel call


first:      ; calculate Ai = i * 2 - 5
    shl eax, 1              ; eax * 2 eq eax << 1
    sub eax, 5              ; eax -= 5

    jmp fillup.endloop
```

```
second:      ; calculate Ai = (i + i / 3) / 2 * 5
    mov dl, 3
    div dl                  ; eax /= 3
    add al, cl              ; al += cl
    shr al, 1               ; eax /= 2 eq eax << 1
    mov dl, 5
    mul dl                  ; eax *= 5

    jmp fillup.endloop

thrid:       ; calculate Ai = (5 * i) / (1 + i)
    mov dl, 5
    mul dl                  ; eax *= 5
    mov dl, cl
    inc dl                  ; dl++
    div dl                  ; eax /= dl

    jmp fillup.endloop
```

2.3. Компиляция и запуск

```
$ make clean run
rm -rf hw1.o hw1
nasm -t -g -O0 -f elf  hw1.asm  -o hw1.o -l hw1.o.lst
gcc -march=i686 -m32  -nostdlibs  -nostartfiles -Wall hw1.o -o hw1
./hw1
ar[1] = -3
ar[2] = -1
ar[3] = 1
ar[4] = 3
. . . .
ar[29] = 4
ar[30] = 4
```

Также был создан дамп массива в файле **dump_asm.bin**

```
$ hexdump dump_asm.bin -C
00000000 fd ff 01 03 05 07 09 0b  0d 0f 23 28 28 2d 32 32
00000010 37 3c 3c 41 04 04 04 04  04 04 04 04 04 04
```

2.4. Проверка

Как и упоминалось ранее, для проверки можно воспользоваться целью **check**, которая компилирует аналогичную программу на си, запускает обе программы, и выводит хэши от файлов с дампами.

2.4.1. Код проверочной программы:

Листинг **hw1.c**

```c
#include <stdio.h>
#include <stdlib.h>

#define COUNT_ELEMENTS 31

int main(void) {
    signed char buf[COUNT_ELEMENTS];
    int i;

    for(i = COUNT_ELEMENTS-1; i >= 0 ; i--) {
        if (i > 20) buf[i] = (5 * i) / (1 + i);
        else if (i > 10) buf[i] = (i + i / 3) / 2 * 5;
        else if (i > 0) buf[i] = i * 2 - 5;
    }

    FILE *of = fopen("./dump_c.bin", "w");
    fwrite(&buf[1], 1, COUNT_ELEMENTS - 1, of);
    fclose(of);

    for(i = 1; i < COUNT_ELEMENTS; i++) {
        printf("ar[%d] = %d\n", i, buf[i]);
    }

    return 0;
}
```

2.4.2. Запуск проверки

```
$ make check
gcc hw1.c -o hw1_c
./hw1_c
ar[1] = -3
ar[2] = -1
. . .
ar[30] = 4
gcc -march=i686 -m32  -nostdlibs  -nostartfiles -Wall hw1.o -o hw1
./hw1
ar[1] = -3
ar[2] = -1
. . .
ar[30] = 4
md5sum dump*.bin
d39df5ece4d29e66c7f36e99547fb733  dump_asm.bin
d39df5ece4d29e66c7f36e99547fb733  dump_c.bin
```

4. Лабораторная работа №2

4.1. Логика работы

Сначала открываются файловые дескрипторы для файлов:

- файл с массивом для обработки(от предыдущего задания) - на чтение
- файл, в который будут помещены элементы из исходного массива, которые соответствуют первому условию - на запись
- аналогично, для второго критерия.

Следующий этап - это чтение массива из файла в заранее отведённую память. На этом инициализация законченна.

Основная цель задания происходит в процедуре **doProcess()**, в которой в цикле перебирается исходный массив, и по критериям задания в зависимости от значения очередного элемента принимается решения в какой массив поместить и нужно ли вообще помещать. Эта процедура использует также 2 процедуры(**firstCriteria()**, **secondCriteria()**) для проверки - попадает ли в критерий значение. Эти процедуры лишь отвечают на вопрос "попал?".

После обработки производится вывод в консоль вызовом процедуры **displayArrays()**, которая использует внутри **displayArray()** - для вывода на экран 1 массива с указанными в аргументах параметрами.

Далее 2 полученных массива записываются на диск в виде файлов с именами **./dump1_asm.bin**, **./dump1_asm.bin**

На этом программа завершает свою работу - закрываются файловые дескрипторы и происходит выход

4.2. Структура программы.

Можно выделить следующие части программы:

- подключение файлов "**hw2_fileio.asm**", "**hw2_proc.asm**" - так же части программы, логически вынесенные отдельно
- объявлении внешних используемых функций(для линкера)
- объявлении данных в секции **.data**
- самой программы (с метки **_start**)
    - открытия 3ёх файлов - **openFileHandles**
    - чтение дампа массива из предыдущего задания - **readDump**
    - разделение массива на 2 по критериям, которые есть в задании - **doProcess**
        - внутри использует функции **firstCriteria** & **secondCriteria** для проверки, подходит ли элемент масива под условие
    - отображение содержимого полученных массивов на stdout - **displayArrays**
        - использует внутри отображение одного массива **displayArray** для каждого
    - запись дампов новых массиов в файлы - **writeDumps**
    - закрытие всех файлов - **closeFileHandles**
    - завершения работы - **exit**:

4.3. Исходный код

4.3.1. Листинг hw2.asm

```
%include "hw2_fileio.asm"
%include "hw2_proc.asm"

extern printf       ; the C function, to be called
extern fwrite
extern fopen
extern fclose
extern fread

section .data
    ar_len:     equ 30
    ar:         times ar_len db 0
    ar1:        times ar_len db 0
```

```
    ar2:        times ar_len db 0
    ar1_indx:   dd 0
    ar2_indx:   dd 0

    dumpfile1:  db './dump1_asm.bin', 0
    dumpfile2:  db './dump2_asm.bin', 0
    dumpfile:   db '../HW1/dump_asm.bin', 0
    dbgline:    db '  ar%d[%d] = %d', 10, 0 ; printf formated string

    ar_txt:     db 'Array #%d', 10, 0

    fofail:     db "Can't open file. Exiting..", 10
    fofail_len: equ $-fofail

    fm_r:       db 'r', 0   ; file mode - 'r'
    fm_w:       db 'w', 0   ; file mode - 'w'

    fh_src:     dd 0    ; file handle - file with source array
    fh_ar1:     dd 0    ; file handle - filtered by 1st criteria
    fh_ar2:     dd 0    ; file handle - filtered by 1nd criteria

    ten:        db 10   ; just 10. Ower numeric base
section .text
    global _start

_start:         ; program entry point

    ; initialize program - open all needed files
    openFileHandles     ; macro

    ; reading array dump into memory
    readDump            ; macro

    ; process array!
    call doProcess      ; proc

    ; display arrays
    call displayArrays  ; proc

    ; write out arrays to files
    writeDumps          ; macro

    ; finalize program - close all file handles
    closeFileHandles    ; macro


exit:   ; exit from program
    mov ebx,0       ; exit code, 0=normal
```

```
    mov eax,1       ; exit command to kernel
    int 0x80        ; interrupt 80 hex, call kernel


cantOpenFile:   ; Error message - can't open file
    mov edx, fofail_len ; Length of error
    mov ecx, fofail     ; Pointer to string
    mov ebx, 1          ; Output to string
    mov eax, 4          ; Sysout command number
    int 0x80            ; Call kernel
    jmp exit            ; Go away :(


displayArrays:  ; just display arrays to stdout
    push dword 1
    push dword [ar1_indx]
    push dword ar1
    call displayArray


    push dword 2
    push dword [ar2_indx]
    push dword ar2
    call displayArray

    ret

displayArray:   ; proc which outputs comment and one array
    push ebp
    mov ebp, esp
    ; arguments (reverse push order):
    ; * [ebp+8] - pointer to array
    ; * [ebp+12] - count of elements in array
    ; * [ebp+16] - array number?

    push dword [ebp+16] ; array number
    push dword ar_txt   ; pointer to string
    call printf         ; call function
    add esp, 2 * 4      ; correction

    mov ebx, [ebp+8]    ; start from here

.loopStart:
    push ebx        ; save current pointer

    xor eax, eax    ; eax = 0
    mov al, [ebx]   ; al = *ebx
    push eax        ; last argument
```

```
    mov eax, [ebp+8]
    sub  ebx, eax    ; current index! :))))
    inc ebx          ; make first element - 1
    push ebx         ; middle number argument -  index
    push dword [ebp+16] ; array number
    push dword dbgline  ; push addr of string - first argument
    call printf
    add esp, 4 * 4   ; normalize..

    pop ebx          ; restore current pointer
    inc ebx          ; set pointer to next element

    mov ecx, ebx     ; loads current element
    sub ecx, [ebp+8]; substract final element number = current index

    cmp ecx, [ebp+12]; compare count and current index
    jne .loopStart   ; no, do iteration again!

    pop ebp
    ret 4 * 3 ; return with stack correction
```

4.3.2. Листинг hw2_fileio.asm

```
%macro openFileHandles 0
    pusha     ; save all registers

    ; open source file
    push dword fm_r          ; file mode - for read
    push dword dumpfile      ; pointer to path
    call fopen               ; call it!
    add esp, 4 * 2           ; correct stack
    mov [fh_src], eax        ; save file handle
    cmp eax, 0               ; is there error?
    je cantOpenFile          ; if error we should break program

    ; open first output file
    push dword fm_w          ; file mode - for write
    push dword dumpfile1     ; pointer to path
    call fopen               ; call it!
    add esp, 4 * 2           ; correct stack
    mov [fh_ar1], eax        ; save file handle
    cmp eax, 0               ; is there error?
    je cantOpenFile          ; if error we should break program

    ; open second output file
    push dword fm_w          ; file mode - for write
```

```
    push dword dumpfile2     ; pointer to path
    call fopen               ; call it!
    add esp, 4 * 2           ; correct stack
    mov [fh_ar2], eax        ; save file handle
    cmp eax, 0               ; is there error?
    je cantOpenFile          ; if error we should break program

    popa      ; restore registers
%endmacro

%macro closeFileHandles 0
    pusha     ; save all registers

    push dword [fh_src]
    call fclose

    push dword [fh_ar1]
    call fclose

    push dword [fh_ar2]
    call fclose

    add esp, 4 * 3  ; correct stack of all calling args
    popa      ; restore registers
%endmacro

%macro readDump 0
    pusha     ; save all registers

    push dword [fh_src] ; file handle
    push dword ar_len   ; block size
    push dword 1        ; size of element
    push dword ar       ; pointer to memory block
    call fread          ; do fread!
    add esp, 4 * 2      ; correct stack

    popa      ; restore registers
%endmacro


%macro writeDumps 0
    pusha     ; save all registers

    push dword [fh_ar1] ; file handle
    push dword [ar1_indx]; count of bytes to write
    push dword 1        ; count of bytes :)
    push dword ar1      ; poiter to first array element
    call fwrite         ; do fwrite!
```

```
    push dword [fh_ar2] ; file handle
    push dword [ar2_indx]; count of bytes to write
    push dword 1        ; count of bytes :)
    push dword ar2      ; poiter to first array element
    call fwrite         ; do fwrite!

    add esp, 4 * 4 * 2  ; correct stack

    popa      ; restore registers
%endmacro
```

4.3.3. Листинг hw2_proc.asm

```
doProcess:  ; Process array
    mov ecx, 0      ; reseting counter

.loopStart: ; start label for inner loop
    xor edx, edx
    mov dl, [ar+ecx]

    push edx            ; push argument - digit
    call firstCriteria  ; check for first criteria
    cmp eax, 0          ; is there no '5'?
    je .loopNext        ; when, test for 2nd criteria

.cc1:
    mov ebx, ar1            ; offset to array
    add ebx, [ar1_indx]     ; add offset to element
    mov [ebx], dl           ; save it!
    inc byte [ar1_indx]     ; index++

.loopNext:
    push edx            ; push argument - digit
    call secondCriteria ; check for first criteria
    cmp eax, 1          ; number is divided by 10?
    jne .loopEnd        ; when, if not - go to end of loop

.cc2:
    mov ebx, ar2            ; offset to array
    mov eax, [ar2_indx]     ; load offeset to element
    add ebx, eax            ; calculate final address
    mov [ebx], dl           ; save it!
    inc byte [ar2_indx]     ; index++

.loopEnd:   ; end of loop
```

```
    inc ecx          ; ecx++
    cmp ecx, ar_len ; it's over?
    jb .loopStart    ; no.. another iteration

.procEnd:
    ret

firstCriteria:  ; is there '5' in number
    mov eax, [esp+4]     ; argument - signed number for check
    test al, (1<<7)      ; is there sign?
    jz .loopStart        ; no, start loop.

    not al         ; so.. remove the minus. first - inverse
    inc al         ; second step to get abs value - +1

.loopStart: ; start looping
    mov ah, 0    ; reset high byte
    cmp al, 0    ; zero?
    jne .continueLoop     ; no. do work.

    mov eax, 0    ; fail.. Not found!
    jmp .procEnd ; return.

.continueLoop:  ; extract next digit
    div byte [ten]    ; divide by 10
    cmp ah, 5         ; is reminder eq 5?
    jne .loopStart    ; no? another iteration.

    mov eax, 1        ; found it!
    jmp .procEnd      ; return!

.procEnd:     ; end of function
    ret 4

secondCriteria:
    mov eax, [esp+4] ; copy arg to eax
    div byte [ten]    ; divide by 10
    cmp ah, 0         ; reminder eq 0?
    je .ok            ; yea! that's ok.

    mov eax, 0        ; no, there reminder > 0
    jmp .procEnd      ; return

.ok:     ; success
    mov eax, 1        ; return true

.procEnd:    ; end of proc
    ret 4    ; correct stack
```

4.4. Компиляция и запуск

```
$ make clean run
rm -rf hw2.o hw2
nasm -t -g -O0 -f elf  hw2.asm  -o hw2.o -l hw2.o.lst
gcc -march=i686 -m32  -nostartfiles -Wall hw2.o -o hw2
./hw2
Array #1
  ar1[1] = 5
  ar1[2] = 15
  ar1[3] = 35
  ar1[4] = 45
  ar1[5] = 50
  ar1[6] = 50
  ar1[7] = 55
  ar1[8] = 65
Array #2
  ar2[1] = 40
  ar2[2] = 40
  ar2[3] = 50
  ar2[4] = 50
  ar2[5] = 60
  ar2[6] = 60
```

В процессе работы также были созданы файлы массивов

```
$ hexdump -C dump1_asm.bin
00000000  05 0f 23 2d 32 32 37 41
$ hexdump -C dump2_asm.bin
00000000  28 28 32 32 3c 3c
```

4.5. Проверка

Вышеприведённый результат работы уже позволяет оценить правильность.

Как и в первой программе, можно проверить результат работы с программой на си, для проверки воспользоваться целью check. Проверка заключается в запуске diff для 2х файлов дампов массивов созданных программой на си и ассемблере.

4.4.1. Код проверочной программы.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define COUNT_ELEMENTS 30


bool firstCondition(signed char number) {
    signed char num = number;
    while(num) {
```

```
        if (num % 10 == 5) return true;
        num /= 10;
    }
    return false;
}

bool secondCondition(signed char number) {
    return (number % 10) == 0;
}

char fileAr1[] = "./dump1_c.bin";
char fileAr2[] = "./dump2_c.bin";

int main(void) {
    signed char buf[COUNT_ELEMENTS];
    signed char ar1[COUNT_ELEMENTS], ar2[COUNT_ELEMENTS];
    int i;
    signed char ch;
    char *ar1_ptr = ar1, *ar2_ptr = ar2, *ptr;
    FILE *outs;

    FILE *srcdat = fopen("../HW1/dump_asm.bin", "rb");
    if (!srcdat) {
        printf("Can't open source array dump for read!");
        return 1;
    }

    for(i = 0; i < COUNT_ELEMENTS && !feof(srcdat); i++) {
        ch = fgetc(srcdat);
        if (firstCondition(ch)) *(ar1_ptr++) = ch;
        if (secondCondition(ch)) *(ar2_ptr++) = ch;
    }

    printf("Array of numbers, which contents '5': \n");

    outs = fopen(fileAr1, "w");
    for(ptr = ar1; ptr < ar1_ptr; ptr++) {
        printf("  ar1[%d] = %d\n", (int)ptr - (int)ar1 + 1, *ptr);
        fputc(*ptr, outs);
    }
    fclose(outs);

    printf("Array of numbers, which can devided by 10: \n");
    outs = fopen(fileAr2, "w");
    for(ptr = ar2; ptr < ar2_ptr; ptr++) {
        printf("  ar2[%d] = %d\n", (int)ptr - (int)ar2 + 1, *ptr);
        fputc(*ptr, outs);
```

```
    }
    fclose(outs);

    return 0;
}
```

4.4.2. Запуск и проверка.

```
$ make check
gcc hw2.c -o hw2_c
./hw2_c
Array of numbers, which contents '5':
  ar1[1] = 5
  . . .
  ar1[8] = 65
Array of numbers, which can devided by 10:
  ar2[1] = 40
  . . .
  ar2[6] = 60
gcc -march=i686 -m32  -nostartfiles -Wall hw2.o -o hw2
./hw2
Array #1
  ar1[1] = 5
  . . .
  ar1[8] = 65
Array #2
  ar2[1] = 40
  . . .
  ar2[6] = 60
Compare 1st dump
diff dump1_asm.bin dump1_c.bin
Compare 2nd dump
diff dump2_asm.bin dump2_c.bin
```

Можно сделать вывод из того, что **diff** ничего не написал - файлы идентичны.